Big Data Anwendungen

Praxisübung - Python

Agenda

- Praxisübung – KNIME
- Praxisübung – Python
  - Grundlagen und Installation
  - Pandas
  - Scikit-learn
  - Übungsaufgaben
- Praxisübung – Freie Aufgabe

Python

- Programmiersprache
  - Gut lesbarer, knapper Code
    (Blöcke über Einrückung, Übersichtliche Syntax)
  - Unterstützung verschiedener Programmierparadigmen
    (Objektorientierung, Aspektorientierung, Funktionale Programmierung)
  - Einsatz als Skriptsprache möglich

- Code Beispiel

```
def fakultaet(x):
    return 1 if x <= 1 else x * fakultaet(x-1)

print ("Hallo Fakultät von 5: {}".format(fakultaet(5)))
```

Python - Installation

- Installation über Paketmanager
  - Python ändert sich stark von Version zu Version
    Konsequenz: Inkompatibilitäten
  - Python besteht aus mehreren Modulen
  - Paketmanager unterstützen bei Umgang

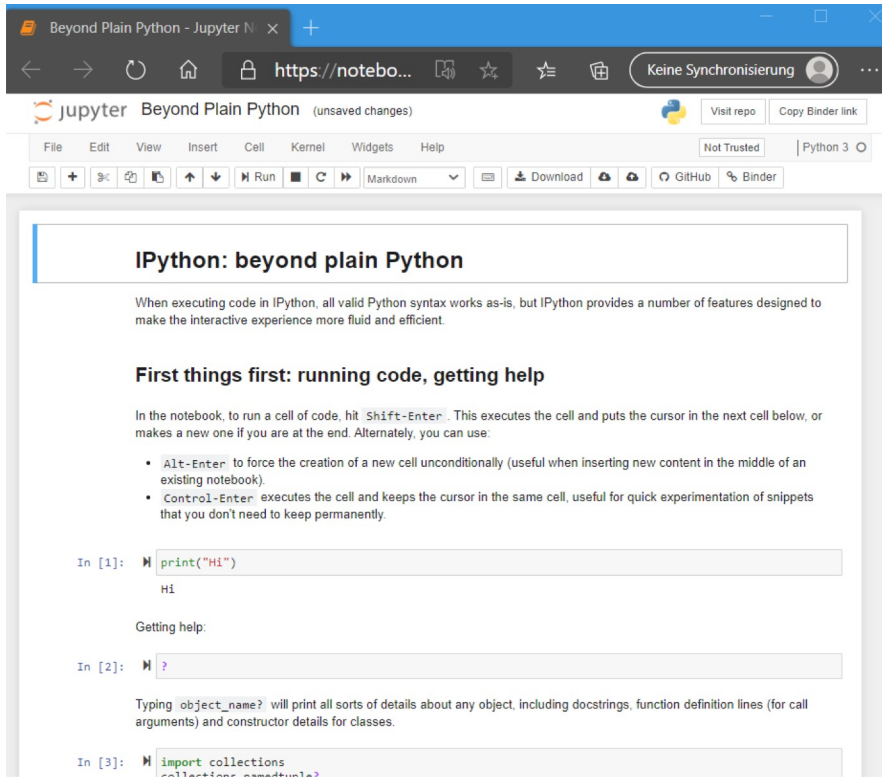
- Paketmanager Windows
  - Anaconda (https://www.continuum.io)
  - Python (x,y) (http://python-xy.github.io/)

- Paketmanager Ubuntu (18.04)

```
sudo apt install python3-numpy python3-scipy
                 python3-pandas python3-sympy
                 python3-nose
```

Jupyter Notebook

- Idee
  - Kombination von „Live“ Quellcode und Ergebnissen
  - Möglichkeit der Kombination mit Formeln, Kommentaren, Bildern
  - Berechnungsergebnisse im Hauptspeicher gespeichert
  - Leichter Austausch von Auswertungen
- Relevante Funktionen für uns
  - Datenbereinigung
  - Statistische Modellierung
  - Visualisierung von Ergebnissen
- Technologie (https://jupyter.org/)
  - Webbassierte Lösung
  - Läuft auch einfach „lokal“
  - Funktioniert auch mit R, Scala, …



Agenda

- Praxisübung – KNIME
- Praxisübung – Python
  - Grundlagen und Installation
  - Pandas
  - Scikit-learn
  - Übungsaufgaben
- Praxisübung – Freie Aufgabe

Pandas (PANel DAta)

- Bibliothek für Python
  - Open-Source Bibliothek von Wes McKinney
  - Entwicklungsbeginn: 2008
  - Unterstützung von Datenmanipulation und -analyse

- Unterstützte Schritte bei der Analyse
  - Preparation
    (inkl. Laden und Manipulieren der Daten)
  - Visualisierung

- Verbreitung
  - Forschung
  - Praxis (Finance, Economics, Statistics, …)

- Key Features
  - Schnell und effizient
  - Datenhaltung im Hauptspeicher
  - Unterstützung von Tabellen und Zeitreihen

Pandas – Datenstrukturen

- Kombination von einzelnen Beobachtungen

Name	Dimensionen	Beschreibung
Series	1	▪ Benanntes, homogenes Array ▪ Unveränderliche Größe
Data Frame	2	▪ Benannte, heterogene Tabelle ▪ Veränderbare Größe
Panel	3	▪ Benannter, heterogener Würfel ▪ Veränderbare Größe

- Eigenschaften
  - Bauen aufeinander auf (z.B. Data Frame ist Menge von Series)
  - Werte in einzelnen Feldern sind veränderlich

- Anmerkungen
  - Data Frame ist sehr populär (nutzen wir hier primär)
  - Panel wird kaum benutzt

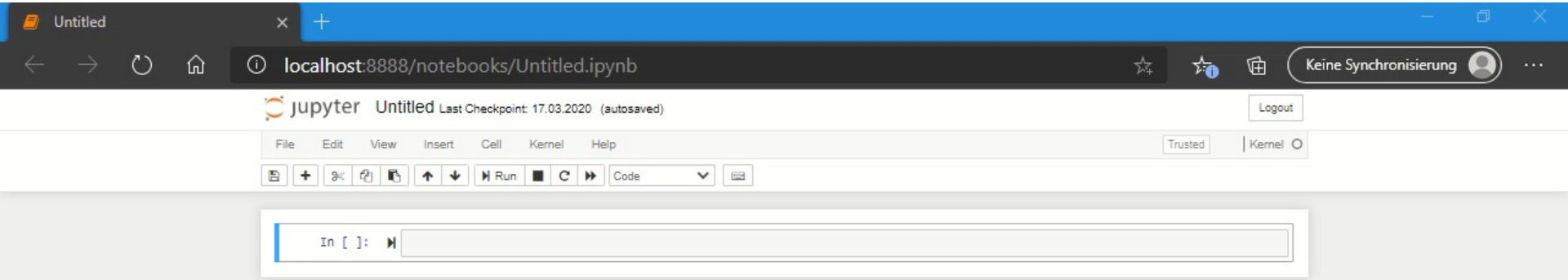
Pandas – Series

- Konstruktor

```
pandas.Series(data, index, dtype, copy)
```

Name	Beschreibung
data	▪ Daten für die Series ▪ Übergabe mittels verschiedener Python Datentypen
index	▪ Benötigt gleiche Länge wie data ▪ Enthält Zugriffsschlüssel für data
dtype	▪ Datentyp der Zellen in data
copy	▪ Gibt an ob Felder aus data kopiert werden sollen ▪ Standard: Falsch

Pandas – Series (Beispiele)



Pandas – Series Zugriff (Beispiele)

- Einzelnes Element abrufen

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d',’e’])
s[3]
```

- Erste drei Elemente abrufen

```
s[:3]
```

- Letzte drei Elemente abrufen

```
s[-3:]
```

- Elemente über Index abrufen

```
s[['c','e']]
```

Pandas – Data Frame

- Konstruktor

```
pandas.DataFrame(data, index, columns, dtype, copy)
```

Name	Beschreibung
data	▪ Daten für die Series ▪ Übergabe mittels verschiedener Python Datentypen
index	▪ Benötigt gleiche Länge wie data ▪ Enthält Namen für Zeilen
columns	▪ Benötigt gleiche Breite wie data ▪ Enthält Namen für Spalten
dtype	▪ Benötigt gleiche Breite wie data ▪ Datentyp der Zellen für jede Spalte
copy	▪ Gibt an ob Felder aus data kopiert werden sollen ▪ Standard: Falsch

Pandas – Data Frame (Beispiele)

- Leeres Data Frame

```
pd.DataFrame()
```

- Data Frame aus Liste

```
data = ['a','b','c','d']
pd.DataFrame(data)
```

- Data Frame aus Liste von Listen

```
data = [['a',1],['b',2],['c',3]]
pd.DataFrame(data,columns=['Name','Wert'])
```

- Data Frame aus Dictionary

```
data = {'a' : [0, 1, 3], 'b' : [1, 4, 5], 'c' : [2, 5, 6]}
pd.DataFrame(data)
```

Pandas – Data Frame Manipulation (Beispiele)

- Data Frame für folgende Schritte

```
data = {'a' : [0, 1, 3], 'b' : [1, 4, 5], 'c' : [2, 5, 6]}
df = pd.DataFrame(data)
```

- Spalte aus bestehender Series einfügen

```
df['d']= pd.Series([10,20,30])
```

- Spalte aus anderen Spalten berechnen

```
df['e']=df['a']+df['d']
```

- Spalte löschen

```
df.pop('d') #Alternative: del df['d']
```

- Zeile löschen

```
df.drop(0)
```

Pandas – Data Frame Zugriff (Beispiele)

- Data Frame für folgende Schritte

```
data = {'a' : [0, 1, 3], 'b' : [1, 4, 5], 'c' : [2, 5, 6]}
df = pd.DataFrame(data, index=['A', 'B', 'C'])
```

- Zugriff auf Spalte

```
df['a']
```

- Zugriff auf Zeile über Zeilenname

```
df.loc['A']
```

- Zugriff auf Zeile über Zeilennummer

```
df.iloc[1]
```

- Mehrere Zeilen abrufen

```
df[1:3]
```

Pandas – Data Frame Basisfunktionen

- Eigenschaften und Funktionen

Funktion	Beschreibung
T	Transponiert Zeilen und Spalten
axes	Gibt die Zeilen- und Spaltennamen zurück
dtypes	Gibt die Datentypen der Spalten zurück
empty	Gibt wahr zurück, wenn Data Frame leer ist
ndim	Gibt Anzahl der Dimensionen zurück
shape	Gibt Anzahl der Zeilen und Spalten zurück
size	Gibt Anzahl der Felder zurück
values	Gibt Werte der Felder zurück
head()	Gibt erste Zeilen zurück
tail()	Gibt letzte Zeilen zurück

Pandas – Aggregate (Beispiele)

- Data Frame für folgende Schritte

```
data = {
   'ID':pd.Series(['A','B','C','D','E','F','G','H']),
   'alter':pd.Series([25,26,25,23,30,29,23,34]),
   'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Summe für alle Spalten

```
df.sum(axis=0) # axis=1: Summe für alle Zeilen
```

- Mittelwert für alle Spalten

```
df.mean(axis=0) # axis=1: Mean für alle Zeilen
```

- Standardabweichung für alle Spalten

```
df.std(axis=0) # axis=1: Standardabweichung für Zeilen
```

Pandas – Zentrale Aggregate

- Wichtige Funktionen

Funktion	Beschreibung
count()	Anzahl Werte (not null)
median()	Median der Werte
mode()	Modalwert der Werte
min()	Minimum der Werte
max()	Maximum der Werte
abs()	Betrag der Werte
prod()	Produkt der Werte
describe()	Ermittelt zentrale Kennzahlen für alle Spalten: ▪ Anzahl ▪ Mittelwert, Standardabweichung ▪ Minimum, Maximum ▪ Quartile

Pandas – Eigene Funktionen (Beispiele)

- Idee
  - Oft eigene, kompliziertere Berechnungen von Werten nötig
  - Funktion wird als „normale“ Pythonfunktion definiert und angewandt
  - Kurzform über „Lambdafunktion“ möglich

- Quadrat eines Werts

```
def quadrat(x):
    return x * x

df['note'].apply(
    quadrat
)
```

- Quadrat eines Werts als Lambdafunktion

```
df['note'].apply(lambda x: x * x)
```

Pandas – Datenbereinigung I (Beispiele)

- Data Frame für folgende Schritte

```
data = {
   'ID':pd.Series(['A','B','C','D','E','F','G','H']),
   'alter':pd.Series([25,26,25,23,30,29,23,34]),
   'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data, index=[0,1,2,3,4,5,6,7,8,9])
df
```

- Umbenennen von Spalten

```
df.rename(columns={'alter' : 'age', 'note' : 'grade'})
```

- Sortieren nach Spalten

```
df.sort_values(by='alter')
```

- Prüfen auf „null“

```
df.isnull()
```

Pandas – Datenbereinigung II (Beispiele)

- Ersetzen einzelner Werte

```
df.replace({34:30,23:25})
```

- Normieren auf vorgegebenes Minimum und Maxmimum

```
df.clip(25,30)
```

- Fehlende Werte mit Konstante ersetzen

```
df.fillna(0)
```

- Zeilen mit fehlenden Werten entfernen

```
df.dropna(axis=0)
```

- Spalten mit fehlenden Werten entfernen

```
df.dropna(axis=1)
```

Pandas – Verbund

- Funktion

```
pd.merge(left, right, how='inner', on=None,
         left_on=None, right_on=None,
         left_index=False, right_index=False,
         sort=True)
```

Name	Beschreibung
left	Linkes Dataframe
right	Rechtes Dataframe
how	Art des Verbunds (später mehr)
on	Attribut für den Verbund
left_on, right_on	Alternative zu „on“, wenn Attributname verschieden
sort	Sortieren des Ergebnisses nach dem Verbundattribut

Pandas – Verbund (Beispiele)

- Data Frames für folgende Schritte

```
left = pd.DataFrame({
   'fach': ['Mathe', 'Statistik', 'VWL', 'BWL', 'BWL'],
   'ort':['HS 1','HS 2','HS 3','HS 4','HS 5']})
right = pd.DataFrame({
   'fach': ['Marketing', 'Statistik', 'VWL', 'BWL'],
   'dozent':['Lichters','Gaffke','Weimann','Spengler']})
```

- Inner Join (nur Tupel erhalten für die left und right Elemente bringen)

```
pd.merge(left, right, on='fach')
```

- Left Join (alle Tupel des linken Data Frame bleiben erhalten – analog right)

```
pd.merge(left, right, how='left')
```

- Outer Join (alle Tupel des linken und rechten Data Frames bleiben erhalten)

```
pd.merge(left, right, how='outer')
```

Pandas – Vereinigung (Beispiele)

- Data Frames für folgende Schritte

```
left = pd.DataFrame({
   'fach': ['Mathe', 'Statistik', 'VWL', 'BWL', 'BWL'],
   'ort':['HS 1','HS 2','HS 3','HS 4','HS 5']})
right = pd.DataFrame({
   'fach': ['Marketing', 'Statistik', 'VWL', 'BWL'],
   'dozent':['Lichters','Gaffke','Weimann','Spengler']})
```

- Vereinigung der beiden Data Frames

```
pd.concat([left, right], sort=False)
```

- Vereinigung unter Beibehaltung des Datenursprungs

```
pd.concat([left, right], sort=False, keys=['left', 'right'])
```

- Vereinigung mit gemeinsamem Index

```
pd.concat([left, right], sort=False, ignore_index=True)
```

Pandas – Liniendiagramm

- Data Frame für folgende Schritte

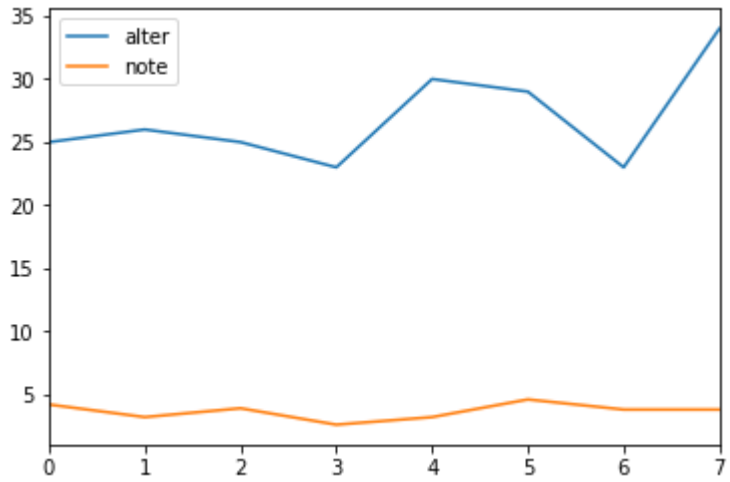
```
data = {
    'ID':pd.Series(['A','A','B','B','C','C','D','D']),
    'alter':pd.Series([25,26,25,23,30,29,23,34]),
    'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Je eine Linie pro numerische Spalte

```
df.plot()
```

- Unter Angabe von x- und y-Werten

```
df.plot(x='alter', y='note')
```



Pandas – Balkendiagramm

- Data Frame für folgende Schritte

```
data = {
    'ID':pd.Series(['A','A','B','B','C','C','D','D']),
    'alter':pd.Series([25,26,25,23,30,29,23,34]),
    'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Normales Balkendiagramm

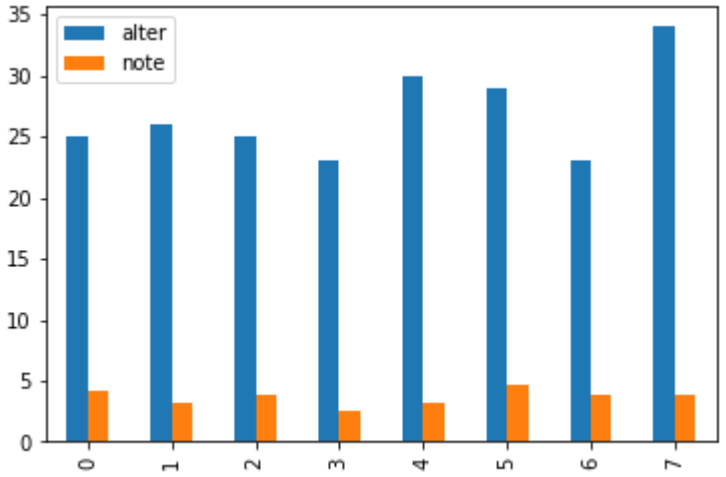
```
df.plot(kind='bar')
```

- Gestapelte Balken

```
df.plot(kind='bar', stacked=True)
```

- Horizontale Balken

```
df.plot(kind='barh', stacked=True)
```



Pandas – Histogramm

- Data Frame für folgende Schritte

```
data = {
    'ID':pd.Series(['A','A','B','B','C','C','D','D']),
    'alter':pd.Series([25,26,25,23,30,29,23,34]),
    'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Normales Histogramm

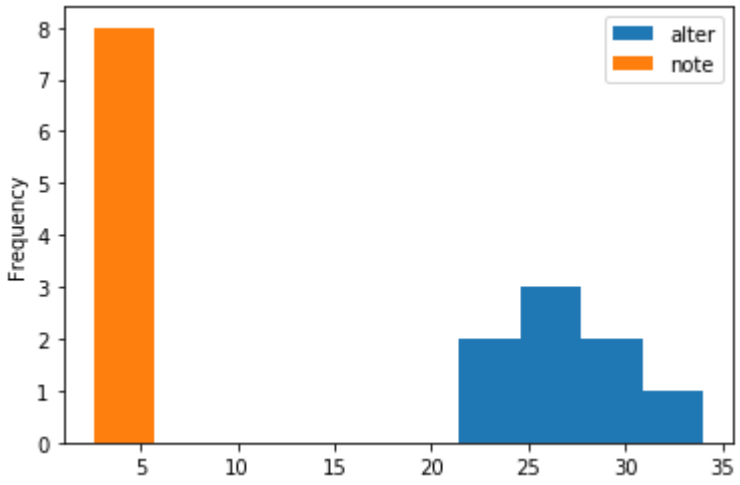
```
df.plot(kind='hist')
```

- Mit vorgegebener Anzahl Bins

```
df.plot(kind='hist', bins=3)
```

- Mit vorgegebenem Binning

```
df.plot(kind='hist', bins=[3,5,10])
```



Pandas – Boxplot

- Data Frame für folgende Schritte

```
data = {
    'ID':pd.Series(['A','A','B','B','C','C','D','D']),
    'alter':pd.Series([25,26,25,23,30,29,23,34]),
    'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Normales Histogramm

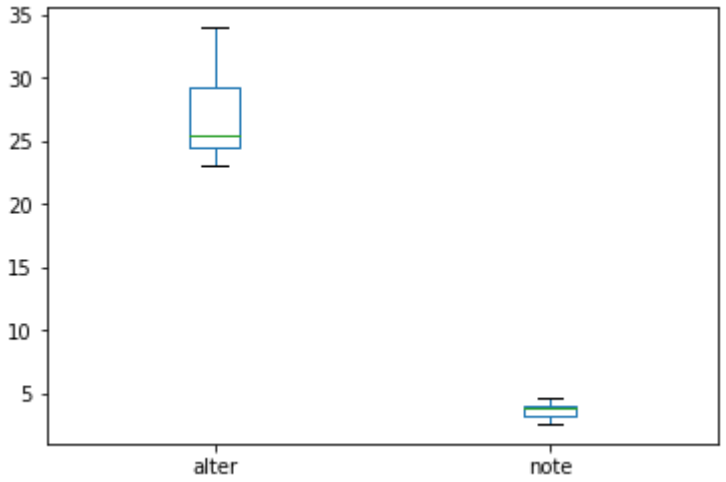
```
df.plot(kind='box')
```

- Nur für ausgewählte Spalten

```
df.boxplot(column='alter')
```

- Gruppiert nach Attribut

```
df.boxplot(column='alter', by='ID')
```



Pandas – Scatterplot

- Data Frame für folgende Schritte

```
data = {
    'ID':pd.Series(['A','A','B','B','C','C','D','D']),
    'alter':pd.Series([25,26,25,23,30,29,23,34]),
    'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Normaler Scatterplot

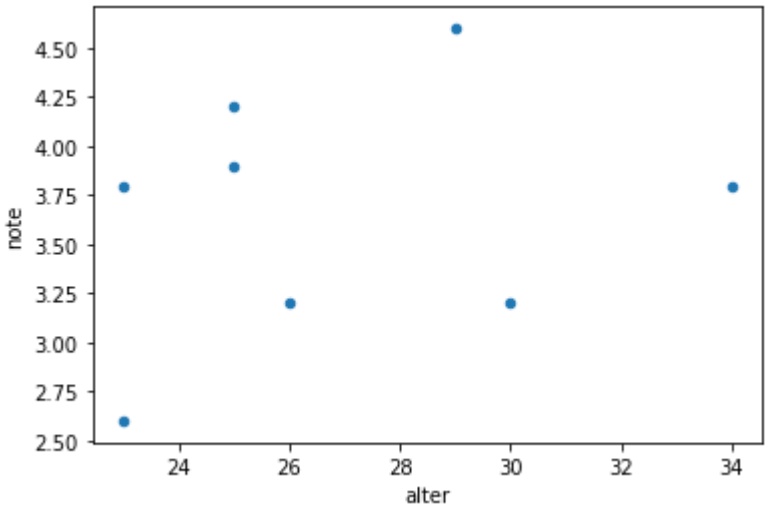
```
df.plot(x='alter', y='note', kind='scatter')
```

- Mit farblicher Kodierung der Punkte

```
df.plot.scatter(x='alter', y='note', c='note')
```

- Mit Kodierung der Größe nach Attribut

```
df.plot.scatter(x='alter', y='note', s=df['alter'])
```



Pandas – Speichern und Laden (Beispiele)

- Data Frame für folgende Schritte

```
data = {'a' : [0, 1, 3], 'b' : [1, 4, 5], 'c' : [2, 5, 6]}
df = pd.DataFrame(data)
```

- Speichern

```
df.to_csv('temp.csv', sep=';', header=True, index=True)
```

- Lesen

```
pd.read_csv('temp.csv', sep=';')
```

- Wichtigste Parameter

Name	Beschreibung
sep, decimal	▪ Zeichen für Trennung der Spalten und Dezimalstellen
header, index	▪ Spalten- bzw. Zeilennamen werden mitgespeichert
quoting	▪ Zeichenketten werden in Anführungszeichen gestellt

Agenda

- Praxisübung – KNIME
- Praxisübung – Python
  - Grundlagen und Installation
  - Pandas
  - Scikit-learn
  - Übungsaufgaben
- Praxisübung – Freie Aufgabe

Scikit-learn

- Bibliothek für Python
  - Open-Source Bibliothek von Wes McKinney
  - Entwicklungsbeginn: 2010
  - Unterstützung von Modellbildung
    (Fokus: Klassifikation, Regression, Clustering)

- Unterstützte Schritte bei der Modellbildung
  - Vorbereitende Schritte (über den Fokus von Pandas hinaus)
    - Skalierung der Daten
    - Imputieren von Daten (nicht nur fixe Werte)
    - Train- Test Splits
  - Modellierungsalagorithmen (auch neue Algorithmen)
  - Bewertung der trainierten Modelle

- Key Features
  - Zusammen mit Pandas sehr mächtig
  - Standardisierte Schnittstelle über Verfahren hinweg (Einarbeitung!)

Scitkit-learn – Einfaches Beispiel

- Laden der Trainingsdaten
  (Hier: Iris-Daten – eignen sich für Klassifikation und Clustering)

```
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
```

- Unterteilen in Training- und Testdaten

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=0
)
```

- Trainieren des Models

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(X_train, y_train)
```

Scikit-learn – Training und Testsplit

- Aufruf

```
sklearn.model_selection.train_test_split(X, y, Parameter)
```

Name	Beschreibung
X	▪ Features von denen Zielgröße abgeleitet werden soll
y	▪ Zielgröße
test_size	▪ Größe der Testdaten in Prozent oder absolut
train_size	▪ Größe der Trainingsdaten in Prozent oder absolut
random_state	▪ Zufallszahl – kann angegeben werden um reproduzierbare Ergebnisse zu erhalten
shuffle	▪ Mischt die Daten vor dem Ziehen
stratify	▪ Attribut für stratified Sampling (vgl. Vorlesung)

Scikit-learn – Training und Testsplit (Beispiel)

- Data Frames für folgende Schritte

```
data = {
   'ID':pd.Series(['A','A','B','B','C','C','D','D']),
   'alter':pd.Series([25,26,25,23,30,29,23,34]),
   'note':pd.Series([4.2,3.2,3.9,2.6,3.2,4.6,3.8,3.8])
}
df = pd.DataFrame(data)
```

- Training und Testdaten aus Features und Zielvariable

```
from sklearn.model_selection import train_test_split
train_test_split(df[['alter', 'note']], df['ID'])
```

- Festlegen der Größe des Testdatensatzes

```
train_test_split(df[['alter', 'note']], df['ID'], test_size=.5)
```

- Durchführen eines stratified Samplings

```
train_test_split(df[['alter', 'note']],
   df['ID'], test_size=4, stratify=df['ID'])
```

Scikit-learn – Entscheidungsbaum

- Aufruf

```
sklearn.tree.DecisionTreeClassifier(Parameter)
```

Name	Beschreibung
criterion	▪ Splitkriterium („gini“ oder „entropy“)
max_depth	▪ Maximale Tiefe des Baums
min_samples_split	▪ Minimale Anzahl Beobachtung für Split (in Prozent oder absolut)
min_samples_leaf	▪ Minimale Anzahl Beobachtung im Blattknoten (in Prozent oder absolut)
random_state	▪ Zufallszahl – kann angegeben werden um reproduzierbare Ergebnisse zu erhalten
class_weight	▪ Anteil der Beobachtungen pro Klasse
max_features	▪ Anzahl der Features die bis Split geprüft werden

Scikit-learn – Entscheidungsbaum (Beispiel)

- Data Frames für folgende Schritte

```
X_train, X_test, y_train, y_test = train_test_split(
    *load_iris(return_X_y=True),
    test_size=0.5, random_state=0
)
```

- Model trainieren

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(
    criterion='gini', splitter='best', max_depth=None,
    min_samples_split=2, min_samples_leaf=1, max_features=None,
    random_state=None, max_leaf_nodes=None, class_weight=None
)
model = dtc.fit(X_train, y_train)
```

- Model evaluieren

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model.predict(X_test))
```

Scikit-learn – Neuronales Netzwerk

- Aufruf

```
sklearn.neural_network.MLPClassifier(Parameter)
```

Name	Beschreibung
hidden_layer_sizes	▪ Anzahl Elemente pro hidden Layer
activation	▪ Art der Aktivierungsfunktion – vgl. Vorlesung (‘identity’, ‘logistic’, ‘tanh’, ‘relu’)
learning_rate	▪ Angabe ob Lernrate konstant (‘constant’) oder mit der Zeif sinkend ist (‘invscaling’)
max_iter	▪ Maximale Anzahl von Iterationen bis Ende
warm_start	▪ Einsatz der letzten Lösung zur Initialisierung des aktuellen Laufs
early_stopping	▪ Algorithmus bricht automatisch ab, wenn sich das Ergebnis nicht signifikant verbessert

Scikit-learn – Neuronales Netzwerk (Beispiel)

- Data Frames für folgende Schritte

```
X_train, X_test, y_train, y_test = train_test_split(
    *load_iris(return_X_y=True))
```

- Model trainieren

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(
    activation='relu', learning_rate='constant',
    max_iter=200, warm_start=False,
    early_stopping=True)
model = mlp.fit(X_train, y_train)
```

- Model evaluieren

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model.predict(X_test))
```

Scikit-learn – Evaluation von Klassifikatoren (Beispiel)

- Akkuratheit

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, model.predict(X_test))
```

- Precision

```
from sklearn.metrics import precision_score
precision_score(y_test, model.predict(X_test))
```

- Recall

```
from sklearn.metrics import recall_score
recall_score(y_test, model.predict(X_test), average='macro')
```

- Jaccard

```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, model.predict(X_test), average='macro')
```

Scikit-learn – KMeans Clustering

- Aufruf

```
sklearn.cluster.KMeans(Parameter)
```

Name	Beschreibung
n_clusters	▪ Anzahl der Cluster
init	▪ Angabe der initialen Clusterzentren oder „random“
max_iter	▪ Maximale Anzahl Läufe bis Algorithmus endet
random_state	▪ Zufallszahl – kann angegeben werden um reproduzierbare Ergebnisse zu erhalten

Scikit-learn – KMeans Clustering (Beispiel)

- Data Frames für folgende Schritte (Kein Train-Testsplit!)

```
X, y = load_iris(return_X_y=True)
```

- Model trainieren

```
from sklearn.cluster import KMeans
kme = KMeans(
    n_clusters = 3, init='random', max_iter=10,
    random_state=3)
model = kme.fit(X)
```

- Model evaluieren (nur bedingt sinnvoll!)

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model.labels_)
```

- Alternative Evaluierung

```
df = pd.DataFrame(X)
df.plot(0, 1, kind='scatter', c=model.labels_)
```

Scikit-learn – Agglomeratives Clustering

- Aufruf

```
sklearn.cluster.AgglomerativeClustering(Parameter)
```

Name	Beschreibung
n_clusters	▪ Anzahl der Cluster
affinity	▪ Distanzmaß für das Clustering (manhattan, euclidean)
compute_full_tree	▪ Gesamten Algorithmus durchführen oder bei n_clusters stoppen
linkage	▪ Kriterium zur Ermittlung der Distanz zwischen zwei Clustern (complete, average, single)

Scikit-learn – Agglomeratives Clustering (Beispiel)

- Data Frames für folgende Schritte (Kein Train-Testsplit!)

```
X, y = load_iris(return_X_y=True)
```

- Model trainieren

```
from sklearn.cluster import AgglomerativeClustering
agc = AgglomerativeClustering(
    n_clusters=3, affinity='manhattan',
    compute_full_tree=False, linkage='single')
model = agc.fit(X)
```

- Model evaluieren (nur bedingt sinnvoll!)

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model.labels_)
```

- Alternative Evaluierung

```
df = pd.DataFrame(X)
df.plot(0, 1, kind='scatter', c=model.labels_)
```

Agenda

- Praxisübung – KNIME
- Praxisübung – Python
  - Grundlagen und Installation
  - Pandas
  - Scikit-learn
  - Übungsaufgaben
- Praxisübung – Freie Aufgabe

Aufgabe (a) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Installieren und öffnen Sie ihr Datenanalyse-Tool.

Aufgabe (b) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Lesen Sie die Daten ein.

Aufgabe (b) - Lösung

- Laden der Daten

```
df = pd.read_csv('data.csv')
```

- Prüfen der Daten auf Plausibilität

```
df.shape
```

- Prüfen der Datentypen

```
df.dtypes
```

- Prüfen der Attributnamen

```
df.columns
```

- Prüfen der Wertebereiche der Attribute

```
df.describe()
```

Aufgabe (c) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Klassifizieren Sie die Daten mit einem Entscheidungsbaum.

Aufgabe (c) – Lösung I

- Die Daten enthalten Beobachtungen ohne Vorhersagen – Entfernen!

```
task = df[df['shot_made_flag'].isnull()]
df = df.dropna()
```

- Zielvariable als y speichern und aus Lerndaten entfernen

```
y = df.pop('shot_made_flag')
```

- Durchführung des Train- Testsplits

```
train_X, test_X, train_y, test_y = train_test_split(df, y)
```

- Identifizieren der Features mit numerischem Wertebereich

```
non_object_features = [
    element for element in train_X.columns
    if (train_X[element].dtype != object)
]
```

Hinweis: Kein Training auf Series vom Typ object möglich

Aufgabe (c) – Lösung II

- Trainieren des Klassifikators

```
dtc = DecisionTreeClassifier(
    criterion='gini', splitter='best', max_depth=None,
min_samples_split=2,
    min_samples_leaf=1, max_features=None,
random_state=None,
    max_leaf_nodes=None, class_weight=None
)
model = dtc.fit(train_X[non_object_features], train_y)
```

- Testen mit der Confusion Matrix

```
confusion_matrix(test_y, model.predict(test_X[non_object_features]))
```

Aufgabe (d) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Identifizieren Sie fehlende Werte und behandeln Sie diese.

Aufgabe (d) – Lösung

- Finden aller Zeilen mit fehlenden Werten

```
df[df.isnull().any(axis=1)]
```

- Keine Zeilen hat fehlende Werte…
  … keine Behandlung notwendig

Aufgabe (e) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Entfernen Sie alle Attribute, die auf Grund ihrer inhaltlichen Bedeutung nicht zum Ergebnis beitragen.

Aufgabe (e) – Lösung

- Betrachten der Spaltennamen

```
df.describe()
```

- Testen mit der Confusion Matrix

```
train_X.pop('game_event_id')
train_X.pop('game_id')
train_X.pop('team_id')
test_X.pop('game_event_id')
test_X.pop('game_id')
test_X.pop('team_id')
```

- Prinzipiell überlegen:
  - Stecken in den Attributen evtl. weitere Informationen (Datum des Spiels / Reihenfolge der Spiele)
  - Solche Informationen finden sich aber in anderen Attributen
  - Damit entfernen OK

- Zudem: Team ID ist immer gleich – kann also raus

Aufgabe (f) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Finden Sie Ausreißer und entfernen Sie diese.

Aufgabe (f) – Lösung

- Betrachten der Kennzahlen zu den Attributen

```
df.describe()
```

- Bei deutlichen Abweichungen des Maximums zum dritten. Quartil…
  … Betrachtung der Verteilungen, z.B.:

```
df['loc_x'].plot(kind='hist')
```

- Ggfs. finden neuer Unter- bzw. Obergrenzen

```
df['loc_y'].quantile(0.95)
```

- Ersetzen der Ausreißer

```
train_X.loc[:,['loc_y']] = train_X['loc_y'].clip(upper=250)
train_X.loc[:,['shot_distance']] =
train_X['shot_distance'].clip(upper=26)
train_X.loc[:,['period']] = train_X['period'].clip(upper=5)
test_X.loc[:,['loc_y']] = test_X['loc_y'].clip(upper=250)
test_X.loc[:,['shot_distance']] =
test_X['shot_distance'].clip(upper=26)
test_X.loc[:,['period']] = test_X['period'].clip(upper=5)
```

Aufgabe (g) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Wandeln Sie die Attribute mit dem Datentyp Objekt so um, dass sie vom Klassifikationsalgorithmus genutzt werden können.

Aufgabe (g) – Lösung

- Shot_type dummyfizieren

```
shot_type = pd.get_dummies(
    test_X['combined_shot_type'], prefix='shot_type')
test_X = test_X.merge(shot_type, on=test_X.index)
shot_type = pd.get_dummies(
    train_X['combined_shot_type'], prefix='shot_type')
train_X = train_X.merge(shot_type, on=train_X.index)
```

Hinweis:
action_type, shot_type ist ähnliche Information nur granularer… hier ignoriert

- Analog zu shot_type noch opponent, shot_zone_area behandlen
  Hinweis:
  Auch hier ähnliche Information in shot_zone_basic und shot_zone_range…

- Heimspiele erkennen

```
df['home'] = df['matchup'].apply(
    lambda x: 0 if x[4:5] == '@' else 1)
```

Aufgabe (h) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Überlegen Sie, ob minutes_remaining und seconds_remaining zu einem sinnvollen gemeinsamen Attribut kombiniert werden kann und führen Sie ggfs. die Kodierung durch.

Aufgabe (h) – Lösung

- Verbleibende Zeit ist Kombination aus beiden Attributen

```
test_X['time_remaining'] =
    test_X['minutes_rem…']*60+test_X['seconds_rem…']
train_X['time_remaining’] =
    train_X['minutes_rem…']*60+ train_X['seconds_rem…']
test_X.pop('minutes_remaining')
test_X.pop('seconds_remaining')
train_X.pop('minutes_remaining')
train_X.pop('seconds_remaining')
```

- Hinweis:
  Wichtig ist hier das Löschen der nicht mehr benötigten Attribute.

- Prinzipiell gilt:
  - Noch einige weitere Optimierungen möglich
  - Ziel kann hier nur erster Überblick sein
  - Bei Interesse: Aufgabe von Kaggel herunterladen und selber ausprobieren!

Aufgabe (i) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Versuchen Sie das bisherige Ergebnis durch Anpassung der Parameter des Entscheidungsbaums zu verbessern.

Aufgabe (i) – Lösung

- Variieren der Parameter – mein bestes Ergebnis:

```
dtc = DecisionTreeClassifier(
    criterion='entropy', splitter='best',
    max_depth=50, min_samples_split=100,
    min_samples_leaf=1, max_features=None,
    random_state=None,
    max_leaf_nodes=None, class_weight={0:0.5, 1:.6}
)
model = dtc.fit(train_X[non_object_features], train_y)
```

- Hinweis:
  Finden der optimalen Parameter muss nicht manuell erfolgen…
  … GridSearch ist Bibliothek für diese Aufgabe

Aufgabe (j) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Versuchen Sie das Ergebnis mit Hilfe einer Support Vector Machine zu verbessern.

Aufgabe (j) – Lösung

- Variieren der Parameter – mein bestes Ergebnis:

```
from sklearn.svm import SVC
svm = SVC(C=1.0, kernel='rbf', degree=3,
    gamma='scale', coef0=0.0, shrinking=True,
    probability=False, tol=0.001, cache_size=200,
    class_weight=None, verbose=False, max_iter=-1,
    decision_function_shape='ovr'
)
model = svm.fit(train_X[non_object_features], train_y)
```

- Hinweis:
  Hier ist sicher noch was drin…
  … auf meinen Notebook läuft die Berechnung aber so lange, dass…
  … ich nach dem ersten Durchlauf abgebrochen habe.

Aufgabe (k) - Aufgabenstellung

- Kobe Bryant beendete seine Basketball Karriere am 12. April 2016. In seinem letzten Spiel für die NBA warf er 60 Punkte für die Los Angeles Lakers. Kobe Bryant begann seine Karriere bei der NBA im Alter von 17 und war einer der erfolgreichsten Basketballspieler der letzten Jahre.

  Der vorliegende Datensatz umfasst 20 Jahre von Daten von Kobe Bryants Treffern und Fehlwürfen. Ihre Aufgabe ist es vorherzusagen, ob ein Wurf ein Treffer oder ein Fehlwurf ist.

  Download: https://www.kaggle.com/c/kobe-bryant-shot-selection

- Versuchen Sie das Ergebnis mit Hilfe eines Neuronalen Netzwerks zu verbessern.

Aufgabe (k) – Lösung

- Variieren der Parameter – mein bestes Ergebnis:

```
mlp = MLPClassifier(activation='identity',
    learning_rate='constant', max_iter=200,
    warm_start=False, early_stopping=True)
model = mlp.fit(train_X[non_object_features], train_y)
```

- Hinweis:
  Finden der optimalen Parameter muss nicht manuell erfolgen…
  … GridSearch ist Bibliothek für diese Aufgabe

Abschließende Worte